Descriptions

- **ALU.v:** I accidentally made the ALU without a 1 bit slice so this was just extra work we did
- **ALU_1bit.v**: Implements a 1-bit arithmetic logic unit (ALU) that can perform basic operations like AND, OR, ADD, and SUB on individual bits. The operations are selected based on control signals (ALUop).
- **Nbit_ALU.v**: The nbit_ALU is a parameterized N-bit Arithmetic Logic Unit (ALU) that performs operations like addition, subtraction, and bitwise logic based on the ALUop signal. It uses individual 1-bit ALU slices (ALU_1bit) to process each bit of the input operands (R2 and R3). The results are stored in the register R1 and clocked into R0 using the nbit_reg module. Carry bits are propagated through the carries bus, and the design ensures synchronized operation with the clock signal.
- **nbit_and.v, nbit_or.v, nbit_not.v**: These modules perform bitwise AND, OR, and NOT operations on N-bit inputs, processing each bit in parallel using 1-bit logic functions.
- **nbit_adder.v**: Implements an N-bit adder, likely using chained 1-bit full adders to perform multi-bit addition.
- **nbit_sub.v**: Implements subtraction for N-bit inputs by adding the two's complement of one operand to another, using an N-bit adder as the backbone.
- **dff.v**: A D flip-flop module used to store a single bit of data, often used in sequential logic to store results between clock cycles.
- **FA_str.v**: A structural Verilog module implementing a 1-bit full adder using basic logic gates (AND, OR, XOR).

Questions

Why use parameters:
the flexibility to use however much bits you want for your alu. Whether its 3 bits or 64 bits it should be able to handle it with the only change being the change in n

How does the sub module works:
it takes in R2 and R3 and it does 2s complement on R3 which it then just goes into the nbit_adder to do normal adding operations with the 2s complement